ITRONオープンセミナー '99

μITRON4.0仕様の概要と μITRON4.0仕様研究会活動報告

1999年6月30日, 7月1日

高田 広章
豊橋技術科学大学／ITRON部会
hiro@ertl.ics.tut.ac.jp

ITRONホームページ
http://www.itron.gr.jp/

ITRONプロジェクト ー 第２フェーズ

第1フェーズ　1984年～

- リアルタイムカーネル仕様の標準化に注力

第2フェーズ　1996年頃～

組込みシステムの
大規模化・複合化

- 周辺仕様まで含めた標準化へ
  - (1) カーネルとの関連での標準化
  - (2) 周辺仕様そのものの標準化

- ソフトウェア部品 (ソフトウェアIP, 実行時ソフトウェア)
  - (1) ソフトウェア部品が流通する前提条件の整備
  - (2) ソフトウェア部品の API の標準化（分野毎）
- 開発環境・言語
- 応用分野に特化した標準化

第２フェーズの標準化の現状

ソフトウェア部品が流通する前提条件の整備

- μITRON4.0仕様 **本日公開**
- アプリケーション設計ガイドライン **検討中**

ソフトウェア部品の API の標準化

- ITRON TCP/IP API仕様 **1998年5月公開**
- JTRON2.0仕様 **1998年10月公開**
- デバイスドライバ設計ガイドライン **検討中**

カーネルとデバッグ環境間のインタフェースの標準化

- μITRON4.0デバッギングインタフェース仕様 **検討中**

C言語以外のプログラミング言語バインディング

- C++言語バインディングの標準化 **今後検討**

応用分野に固有の要求への対応

- RTOS自動車応用技術委員会 **完了** → μITRON4.0仕様

μITRON4.0仕様 ー 策定の必要性

ソフトウェアの移植性の向上

- 組込みソフトウェアの大規模化により移植性が重視
- 移植性の向上はソフトウェア部品流通の前提条件

ソフトウェア部品向け機能の追加

- 外販することを前提としたソフトウェア部品開発

新しい要求・検討成果の反映

- リアルタイム性の保証を容易にするための機構
  - ← リアルタイム性を持ったソフトウェア部品
- よりコンパクトな実装を可能にする仕様
  - ← 自動車制御応用における要求事項の整理

半導体技術の進歩への対応

- μITRON3.0を公開してから約6年が経過

ソフトウェアの移植性の向上

- 基本的には標準化の度合いを強くすればよい

一般には両立は困難

- 適応化の利点（弱い標準化によって得られる）
  - 広範なスケーラビリティの実現
  - ハードウェア（プロセッサ）の特徴を活かす
  - アプリケーション毎の要求への対応

両立可能な部分は両立させる

移植性の向上

- 不必要な実装依存性の削減
  - ! 「決める理由がなければ決めない」から「決めない理由がなければ決める」へ転換

スケーラビリティの向上

- 使わない機能が簡単に取り外せればよい
  ただし，ライブラリリンクによる取り外しを前提に
  ← テストの問題
  ※ 条件コンパイルによる再構成（*cf.* eCos）
- 同期・通信オブジェクトのタスク独立化
- システムコールの単機能化

↓

このアプローチが有効に働く範囲では，豊富な機能を提供して，必要なものだけ組み込む方法が有利

! *スケジューラ/ディスパッチャに関わる部分は取り外し困難*

↓ **例）**タスク例外処理，ミューテックス

- RISC的なリアルタイムカーネル仕様
- プロファイル規定の導入

スタンダードプロファイルの導入

コンセプト

- μITRONの適用分野の中で比較的大規模なシステムを想定し，*標準的な機能セットを「強く」標準化*
  - ← 移植性が重視されるのは比較的大規模なシステム
- 性能重視の小規模なシステムにはサブセットで対応
- 標準を越える要求のために拡張機能も定義

μITRON4.0仕様全体	… 弱い標準化
スタンダードプロファイル	… 強い標準化

言い換えると…

- 移植性を重視するソフトウェア（典型例: ソフトウェア部品）はスタンダードプロファイルの機能のみを用いる
- ソフトウェアの移植性を重視する分野向けのカーネルはスタンダードプロファイルに準拠して実装する

想定するシステムイメージ

- ハイエンド 16bit ～ 32bit プロセッサ
- カーネルサイズ 10～20KB程度（全機能を使った場合）
- *システム全体が1つのモジュールにリンク*
- *カーネルオブジェクトは静的に生成*

規定の概略

- サポートすべきサービスコールとその機能の規定
  - μITRON3.0仕様のレベルSのほとんど（一部仕様修正）
  - μITRON3.0仕様のレベルEの一部（一部仕様修正）
  - 新規の機能（データキュー，例外処理のための機能，システム状態参照機能など）
- 割込みハンドラから呼び出せるサービスコールの規定
- コンフィギュレーション方法の標準化（静的API）
- その他の規定についても実装依存性を削減

コンフィギュレーション方法の標準化

- カーネルオブジェクトは静的に生成
  - → オブジェクト生成情報の，システムンフィギュレーションファイル中での記述方法を標準化

静的API（*static API*）

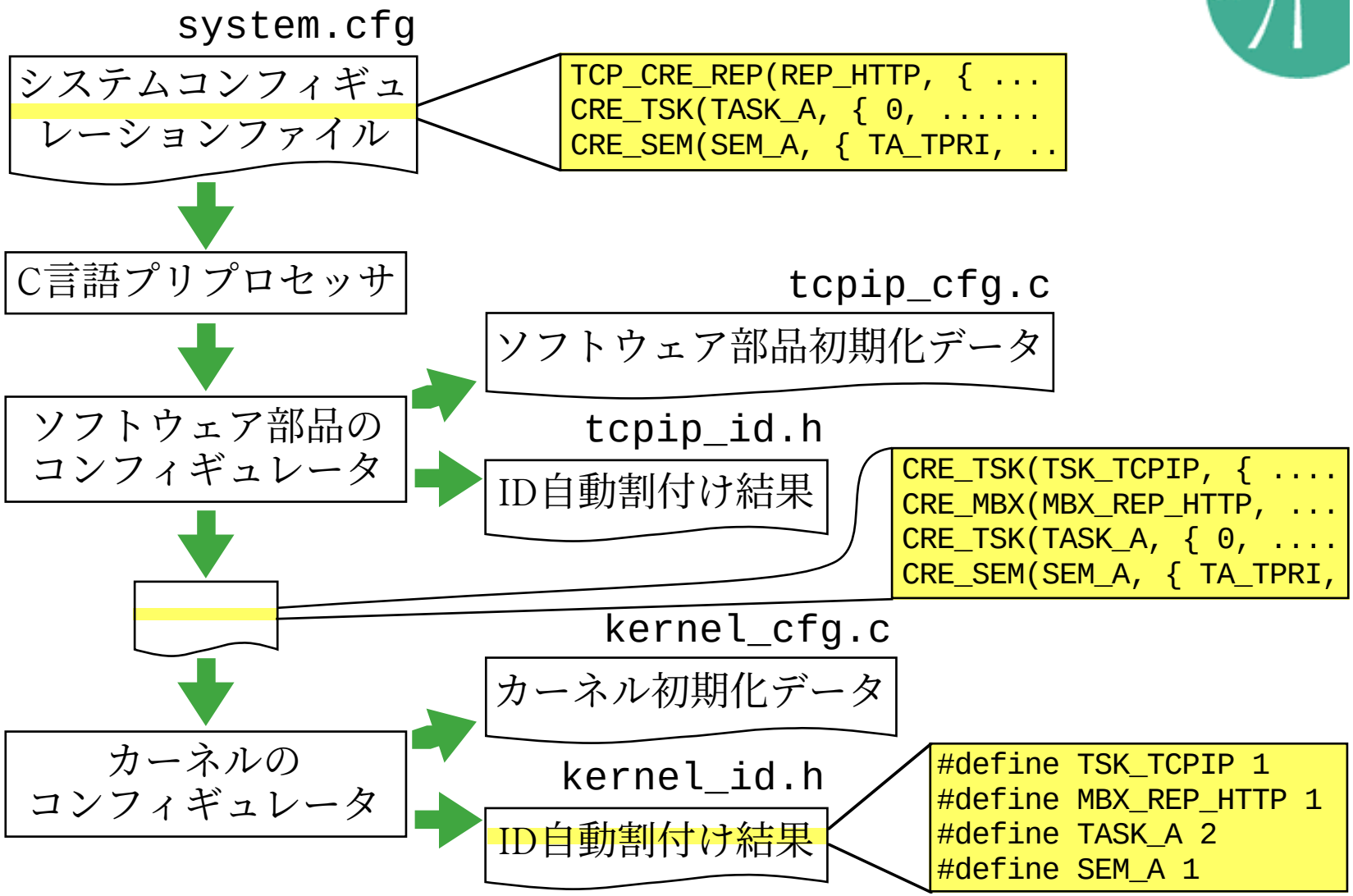
例）
```
CRE_TSK(tskid, { tskatr, exinf, task,
                 itskpri, stksz, stk });
ATT_ISR({ isratr, exinf, intsrc, isr });
```

- ソフトウェア部品の静的APIを混在して記述可能

利点

- ソフトウェアを別のカーネルへ移植するのが楽に
- ID番号の割り付ける作業を自動化
- ソフトウェア部品の組み込みが容易に
- サービスコールと静的APIを個別に覚える必要がない

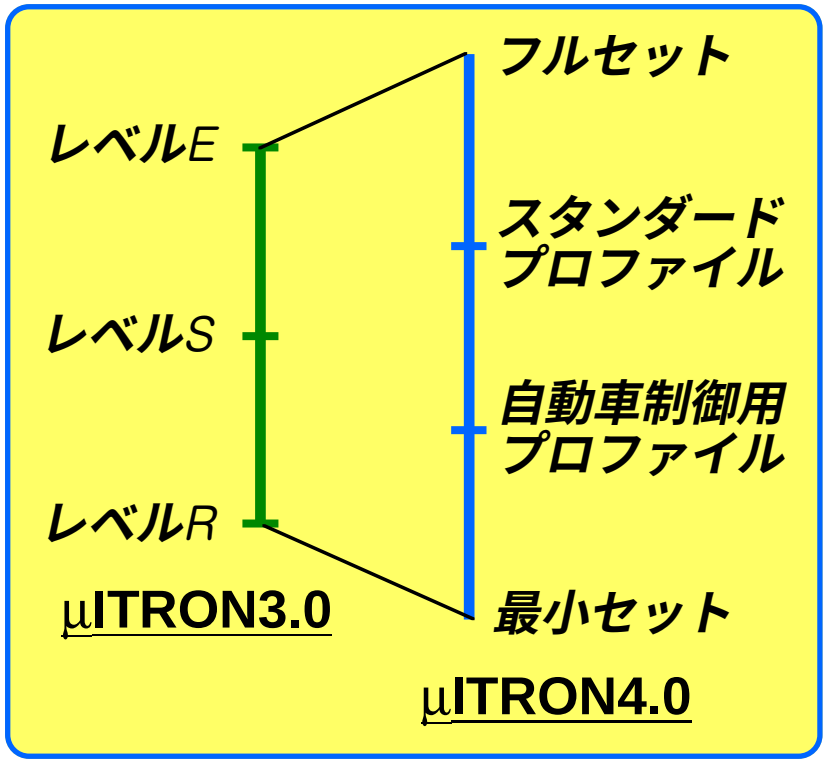
コンフィギュレーション手順



より広いスケーラビリティの実現

μITRON3.0仕様よりも高機能化

- データキュー
- タスク例外処理機能
- システム状態参照機能
- 割込みサービスルーチン
- ハードリアルタイム対応
- ID番号の自動割付け



自動車制御用プロファイル

- 小規模なシステム向けのプロファイル規定

より小規模なシステムへも適用可能に

- 待ち状態をオプションにして，休止状態を必須に

自動車制御用プロファイルの概要

- RTOSの必要性が高まっているにも関わらず，その適用が難しかった分野 → RTOS自動車応用技術委員会

スタンダードプロファイルから削減する機能

- 強制待ち状態，タスク例外処理機能
- 優先度順の待ち行列，タイムアウト付きのサービスコール
- メールボックス，固定長メモリプール　など

制約タスク → 使用メモリ節約のため

- 待ち状態に入れないタスク
- 複数の制約タスクでスタックエリアを共有可能
- 待ち状態に入ろうとした時にエラーが報告されることに依存しなければ，通常のタスクにしてもそのまま動作

! この意味でスタンダードプロファイルの下位互換

プロファイルの機能レベル

サービスコール/静的APIの数

	サービスコール	静的API
フルセット	166（13※）	21
スタンダードプロファイル	70（13※）	11
自動車制御用プロファイル	43（11※）	8

※ かっこ内は非タスクコンテキスト専用サービスコール（内数）

参考）μITRON3.0仕様のシステムコールの数

レベルR: 11　　レベルR+S+E: 95

レベルR+S: 35　　レベルR+S+E+C: 103

（タスク独立部専用のシステムコールは含まない）

実装する上での各プロファイルの特徴

スタンダードプロファイル

- 一つのサービスコールの発行で複数のタスクが待ち解除されることはない
- 優先度順の待ち行列／メッセージキューはある
- カーネル内での動的なメモリ管理は必要ない
- (頑張れば) 静的な情報の活用が可能

自動車制御用プロファイル

- 待ち行列はすべてFIFO順
- タイマキューで管理するのは周期ハンドラのみ
- (うまく作れば) サービスコールの最大実行時間をタスク数に依存せずに押さえられる（周期ハンドラの数には依存）

μITRON4.0仕様の機能概要

- タスク管理機能
- タスク付属同期機能
- タスク例外処理機能
- 同期・通信機能
  セマフォ, イベントフラグ, データキュー, イベントフラグ
- 拡張同期・通信機能
  ミューテックス, メッセージバッファ, ランデブ
- メモリプール機能
- 時間管理機能
  システム時刻管理, 周期／アラーム／オーバーランハンドラ
- システム状態管理機能　　システム状態参照機能
- 割込み管理機能　　割込みサービスルーチン
- サービスコール管理機能
- システム構成管理機能

μITRON4.0仕様の新機能

- データキュー
- タスク例外処理機能
- システム状態参照機能

} スタンダードプロファイル内

- 割込みサービスルーチン
- ハードリアルタイム対応
  - ミューテックス（優先度継承，優先度上限）
  - オーバーランハンドラ
- ID番号自動割付け

  オブジェクト生成時にID番号を自動割付けする機能

  例）`tskid = acre_tsk(&pk_ctsk);`

  `tskid` ← 生成したタスクのID番号が返る

  `&pk_ctsk` ← タスク生成情報

データキュー

機能の概要

- 1ワードデータのメッセージ通信機構．リングバッファで実装することを想定（μITRON3.0仕様のリングバッファで実装されたメールボックスに相当）
- 自動車制御応用から強い要求，他の応用でも有用
- 使わない場合にはリンクされない

データキューのAPI

CRE_DTQ … 生成（静的API）
rcv_dtq, prcv_dtq, trcv_dtq … 受信（待ちあり）
snd_dtq, psnd_dtq, tsnd_dtq … 送信（待ちあり）
fsnd_dtq … バッファフルの時に最も古いデータを上書き
isnd_dtq, ifsnd_dtq … 割込みハンドラからも送信可能
cre_dtq, del_dtq, ref_dtq … 生成・削除・参照（拡張機能）

例外処理のための機能

例外処理のモデル

CPU例外ハンドラ ← 例外発生時にプロセッサが起動

（CPU例外の種別毎にアプリケーションで定義）

↓ 必要に応じて例外処理をタスクに任せる

タスク例外処理ルーチン

（タスク毎にアプリケーションで定義）

- タスク例外処理ルーチン

  UNIXのシグナルハンドラを軽くしたような機能

  - サービスコールにより明示的に例外処理を要求
  - タスクが次にスケジュールされる時にルーチンを呼び出す
  - タスクと同じコンテキストで実行
  - ルーチン起動時にタスク例外処理禁止状態に
  - タスク毎に１つのみで，例外要因はパラメータで渡す

例外処理のためのAPI

- CPU例外ハンドラ

  DEF_EXC／def_exc … CPU例外ハンドラ定義

- タスク例外処理機能

  DEF_TEX … タスク例外処理ルーチン定義（静的API）
  ras_tex … タスク例外処理要求
  iras_tex … 割込みハンドラからも例外処理要求可能
  dis_tex … タスク例外処理禁止
  ena_tex … タスク例外処理許可
  def_tex，ref_tex … 定義，参照（拡張機能）

- 例外要因は ras_tex にパラメータとして渡すと，タスク例外処理ルーチンに渡される．複数の ras_tex が発行された時には，例外要因をビット毎論理和を取る
- タスクの拡張情報もタスク例外処理ルーチンに渡される

システム状態参照機能

導入の動機

- 外販するソフトウェア部品では，サービスコールを呼び出すコンテキストを限定するのは難しい
  - → 正しくないコンテキストから呼ばれたらエラーを返す

! μITRON3.0では，現在のコンテキストを調べる機能がオーバヘッドの大きい拡張機能でしか用意されていない

システム状態参照のためのAPI

sns_ctx … 割込みハンドラ実行中か？
sns_loc … CPUロック状態か？
sns_dsp … ディスパッチ禁止状態か？
sns_dpn … ディスパッチ保留状態か？（待ちに入れるか？）
sns_tex … タスク例外処理禁止状態か？

- どこからでも呼び出し可能で，必ずブール値を返す

CPUロック状態の意味の変更

- CPUロック状態とディスパッチ禁止状態は独立に
  - → ディスパッチ禁止中に呼ばれたソフトウェア部品でCPUロックした場合でも，元に戻すことが可能に

例)

```
{
        BOOL locked = sns_loc();
        if (!locked) loc_cpu();
        排他制御が必要な処理
        if (!locked) unl_cpu();
}
```

- 割込みハンドラ中でもCPUロックできる
  - → 割込みハンドラ中で多重割込みを禁止するためのポータブルな方法を提供
- CPUロック状態では，いくつか例外の除いてはサービスコールを呼び出すことができない

割込み処理モデル

動機

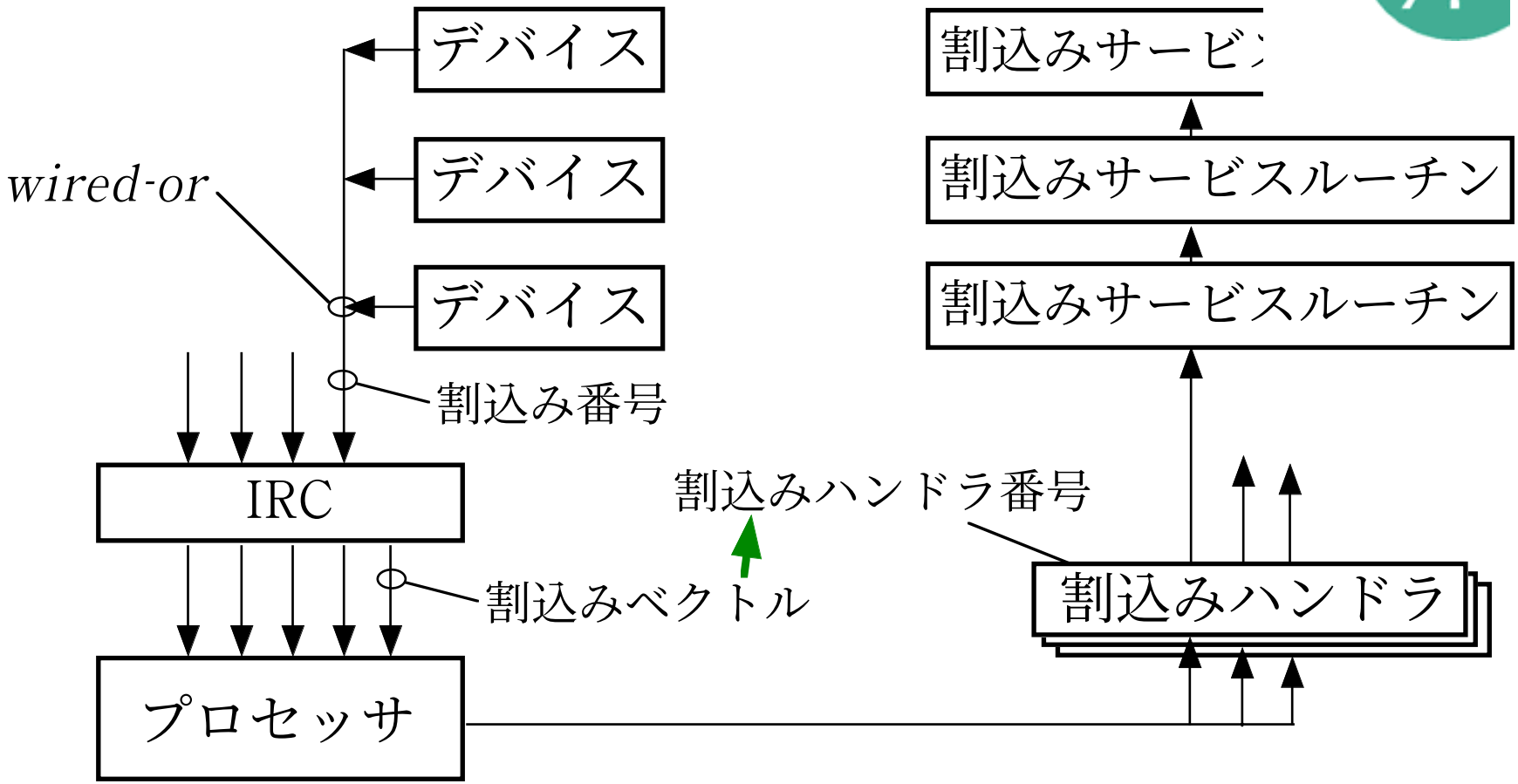
- ポータブルな割込み処理の記述方法を提供したい
  - 異なるプロセッサ, システム, OS へのポータビリティ
  - ! 割込み発生源のデバイスのみに依存して記述

※ カーネルがIRC（割込みコントローラ）に依存するのは避けたい場合も

２階層の割込み処理モデル

- 割込みハンドラ（INH）
  - 低レベルな割込み処理記述の方法
- 割込みサービスルーチン（ISR）
  - ポータブルな（抽象度が高い）割込み処理記述の方法

ハードウェアモデルとソフトウェアモデル



ハードウェアモデル

ソフトウェアモデル

割込みハンドラ（INH）

- プロセッサの機能のみに依存して起動するのが基本
- 割込みコントローラ（IRC）の操作が必要なら，割込みハンドラ内で行う（カーネルでは行わないのが基本）
- そのままの形で異なるシステムに移植することは不可能
- 割込みハンドラ番号（INHNO）で識別．割込みハンドラ番号は，一般的な実装ではベクトル番号に対応
- DEF_INH，def_inh により登録
- (必要なら) 出入口処理をカーネルで用意

割込みサービスルーチン（ISR）

- プロセッサやIRCに依存せずに記述可能．割込み発生源のデバイスにのみ依存
- 割込みハンドラから起動（IRCの操作は割込みハンドラ側で行われる）．割込みハンドラはカーネルで用意
- 割込み番号（INTNO）で対象となる割込みを指定．割込み番号は，一般的な実装ではIRCへの入力ラインに対応
- 割込みサービスルーチンは，デバイスからの割込み出力に対応．同一の割込み番号に対して複数の割込みサービスルーチンを登録可能．すべてを順に起動
- 割込みサービスルーチンIDで識別
- ATT_ISR，cre_isr により登録
- ポータブルに記述するためのガイドラインは，デバイスドライバ設計ガイドラインWGで検討中

ミューテックス

- 優先度継承/優先度上限プロトコルをサポートする排他制御機構
  - → POSIX リアルタイム拡張の mutex に相当
- 上限のない優先度逆転を防止
- 最大資源数が1のセマフォとの他の違い
  - ロックしたタスク以外はロック解除できない
  - タスク終了時に自動的にロック解除される

オーバーランハンドラ

- タスクが設定されたプロセッサ時間を使い切った時に例外を発生させる機能
- オーバーランハンドラはシステムに一つのみ登録できる
- プロセッサ時間の測定精度は実装依存

μITRON3.0からの主な変更

サービスコール名の変更

- preq_sem → pol_sem
- snd_msg → snd_mbx，rcv_msg → rcv_mbx
  get_blf → get_mpf，rel_blf → rel_mpf
  get_blk → get_mpl，rel_blk → rel_mpl
- get_ver → ref_ver，ref_i*XX* → get_i*xx*

タスクの起動

- 起動要求のキューイングできる act_tsk を標準に
- 起動コードの渡せる sta_tsk は拡張機能に

タスク/ハンドラの記述方法

- C言語からの ret_int は廃止．関数からリターンで割込みハンドラからリターン（他のハンドラも同様）
- タスクのメイン関数からのリターンでもタスク終了

タイムイベントハンドラ

- 周期ハンドラ
  - CRE_CYC … 定義（静的API），周期・位相を与える
  - sta_cyc … 周期ハンドラの動作開始
  - stp_cyc … 周期ハンドラの動作停止
- アラームハンドラ
  - CRE_ALM … 定義（静的API），時間は与えない
  - sta_alm … アラームハンドラの動作開始，時間を与える
  - stp_alm … アラームハンドラの動作停止

その他

- 自タスクに対する sus_tsk を可能に
- タスク優先度の最低段数は16段階
- ランデブ呼出し待ちのタイムアウトの意味の変更
- 実装依存性の削減・仕様の厳密化

などなど

システムの初期化手順

- システム初期化の流れ

  ハードウェア依存の初期化処理 ← リセット

  （ユーザが作成ÅCカーネルの管理外）

  ↓

  カーネル自身の初期化

  ↓

  静的APIの処理，*初期化ルーチンの実行*

  （ユーザが作成，静的APIで定義）

  ↓

  カーネル起動，タスクの実行が開始

  割込みが許可，システムクロックが動作開始

- 初期化ルーチンでは割込みは禁止．呼び出せるサービスコールは実装定義
- 静的APIでのエラーの処理は実装定義

μITRON4.0仕様書の構成

第１章　μITRON4.0仕様策定の背景

第２章　ITRON仕様共通規定

- μITRON4.0仕様とそれと整合するように標準化されるソフトウェア部品仕様に共通の規定

第３章　μITRON4.0仕様の概念と共通定義

- 複数の機能単位にまたがる概念や共通の定義

第４章　μITRON4.0仕様の機能

- サービスコールと静的APIの機能（機能単位毎）

第５章　付属規定

- 仕様準拠の条件，自動車制御用プロファイル など

第６章　付録

第７章　リファレンス

※　　で示したものは仕様の本体

仕様書の作成方針

- 仕様の本体では，わかりやすさよりも厳密性を重視
  ！移植性向上のために必要

  例）「実装定義」「実装依存」「未定義」「実装独自」を使い分け

  ↓

  わかり易い解説書（標準ガイドブック）は別途作成予定
- スタンダードプロファイルの規定は機能仕様毎に記述.
  自動車制御用プロファイルは付属規定でまとめて記述
- 誤解のおそれがある箇所や記述が分散していて意図が読み取りにくい箇所には，補足説明を入れる
- μITRON3.0仕様との違いも明記する
- 完成した仕様を読んだだけでは背景がわからない箇所には，仕様決定の理由を入れる

μITRON4.0仕様研究会

ハードリアルタイム
サポート研究会

RTOS自動車応用技術委員会

μITRON4.0仕様研究会

- カーネル仕様WG
- デバッギングインタフェース仕様WG
- アプリケーション設計ガイドラインWG
- デバイスドライバ設計ガイドラインWG

↔ TCP/IP API

↔ JTRON

- オープンな研究会（誰でも参加可能）
- トロン協会 ITRON部会が主催
- 1998年4月に活動を開始（デバッグWGは1999年2月より）

デバッグ環境とのインタフェースの標準化

現状の問題点

- ! デバッグ環境（デバッガ, ICE, ロジアナなど）はそれぞれのμITRON仕様カーネルに個別に対応することが必要

デバッギングインタフェースの標準化

- μITRON仕様のリアルタイムカーネルとデバッグ環境のRTOS**サポート機能**とのインタフェースを標準化

標準化の利点

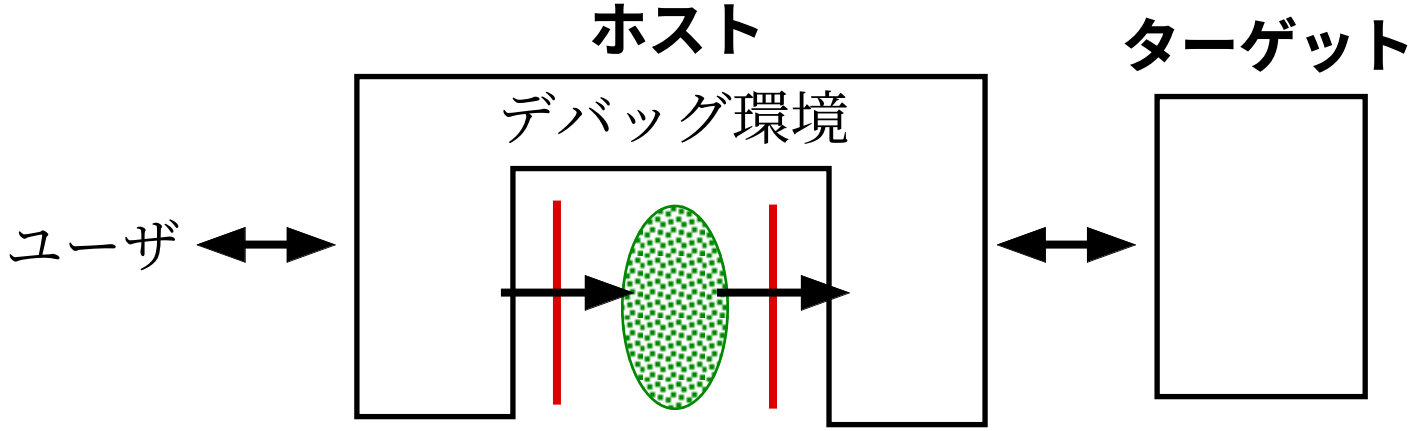
- デバッグ環境が，多種のカーネルへ対応することが容易に
- カーネル側は，開発環境を充実させることが容易に

標準化の目標

- カーネルのオーバヘッドが最小限
- 異なるデバッグ環境とのインタフェースをできる限り共通化
- 他のRTOSやソフトウェア部品にも適用可能

アプローチ

- デバッグ環境側からの**処理要求**を標準化
  例) あるタスク/オブジェクトの状態を読み出す
- デバッグ環境における処理の**プリミティブ**を標準化
  例) 指定した番地のメモリの内容を読み出す
- RTOSインタフェースモジュール
  あるカーネルで，デバッグ環境側からの処理要求を，デバッグ環境におけるプリミティブを使って処理する方法を，カーネルメーカ側が**プログラムの形**で提供



デバイスドライバ設計ガイドラインWG

Device Interface Component (DIC)

- 「デバイスドライバ」という用語は人によってイメージが異なるので避けた方が無難
- ハードウェアデバイスを扱うための最低限のインタフェースソフトウェア．OS のための抽象化は提供しない
- アプリケーションプログラマが直接呼び出す
- デバイスメーカが提供してくれると理想的

→ *ポータビリティの確保が極めて重要*

DIC の階層モデル

- Primitive DIC ＝ 最下位層のDIC
  - OSに依存せずに記述するガイドライン
- General DIC ＝ 上位層のDIC

まとめと今後のプラン

- 組込みシステム開発のオープン化の流れ

  既製の技術を安心して使うには標準化が不可欠

↓

μITRON4.0仕様はオープン化の流れに対応する基盤

今後の計画

- C/EC++ API の標準化
- 検定仕様書の策定
- ITRON TCP/IP API仕様，JTRON仕様を整合させる

μITRON4.0仕様の仕様書・最新情報

ITRONホームページ

http://www.itron.gr.jp/